Chapter 12 Contributions and Future Research

Software engineering researchers are investigating approaches to automate software design methods that were previously developed for use by designers. Most investigations to date propose approaches to automate the Structured Design method described by Yourdon and Constantine. These investigations are motivated primarily by the fact that Structured Design is a well-known method, is used by many practicing software designers, and is supported by numerous computer-aided software engineering (CASE) tools. Unfortunately, the structured design method leads only to sequential designs. A larger class of software systems, including concurrent and real-time designs, is not addressed by Structured Design.

12.1 Contributions

The research described in this dissertation proposes an approach leading to automated generation of concurrent designs for real-time software, given a data/control flow diagram model of the problem. The approach consists of two main parts: 1) a means of analyzing a data/control flow diagram and interpreting elements on the diagram as semantic concepts in a specification meta-model and 2) a means of generating concurrent designs from a specification meta-model. This dissertation contains a complete specification of all components of the approach. This dissertation also describes the implementation and application of a prototype COncurrent Designer's

Assistant, or CODA, that is built directly from the specifications contained in this dissertation. The detailed contributions of the research described in this dissertation are as follows.

- A specification meta-model is defined that enables semi-automated inference of the existence of semantic concepts from a data/control flow diagram that is described using RTSA notation. This specification meta-model begins with ideas expressed by Gomaa when describing COBRA, a method which applies RTSA notation in a restricted form to model the functional and behavioral characteristics of a software system. This dissertation provides a concept hierarchy, a set of classification rules, and a set of concept axioms that extend and formalize the ideas contained in COBRA.
- A design meta-model is defined that enables concurrent designs to be represented and reasoned about. The design meta-model allows a design to be viewed as an object-oriented database that can be queried interactively by a designer. The design meta-model also enables automated assessment of the completeness of a design, with respect to the input specification, and automatic checking of generated designs for consistency with the design meta-model. The design meta-model includes bi-directional traceability between a concurrent design and the data/control flow diagram from which the design is generated. The design meta-model provides for design histories to be associated with each component in a concurrent design. The dissertation also illustrates and describes a means of representing diagrammatically

most of the entities and relationships included in the design meta-model. In addition, a means is identified to model some salient characteristics of operating system services, hardware configurations, and design parameters that can influence design decisions.

- The design heuristics embodied in the CODARTS design method are specified as expert-system rules that reason from the specification and design meta-models. The rules are grouped together into decision-making processes allocated to each of four phases in the CODARTS design method. In the case of conflicting rules, preferred rule orderings are specified. Consultation with the designer is limited to cases where additional insight can improve a design decision and where an experienced designer is available.
- The specification and design meta-models, the decision-making processes and related control knowledge, and the design-decision rules are implemented in a prototype COncurrent Designer's Assistant, or CODA. CODA is applied to generate, semi-automatically, concurrent designs for four real-time problems. For each problem, the design produced by CODA is compared against an existing design provided by an experienced designer. The effectiveness of the approach is evaluated.

12.2 Potential Applications

The results from this research might be applied to assist designers to create concurrent designs. In one application, a tool such as CODA might be embedded into a computer-aided software engineering (CASE) system. Most CASE systems enable a designer to enter flow diagrams and structure charts, or other representations of a software design; however, the process of creating the software design from the flow diagrams must be performed by a human designer, outside the CASE system, without automated assistance. Where a tool such as CODA is available, a designer could enter a data/control flow diagram into a CASE system and then invoke automated assistance to generate a concurrent design. Such automation can capture and maintain traceability between elements on the data/control flow diagram and components of the concurrent design. In addition, such automation can capture and report design decisions and rationale.

In a second application, a tool such as CODA might be applied as a training aid to help develop the skills and understanding of software designers with respect to two, complementary software design methods, COBRA and CODARTS. Problems could be assigned to students who would then use COBRA to develop a data/control flow diagram and go on to construct a concurrent design using the CODARTS heuristics. The data/control flow diagram could then be entered into a tool such as CODA. The concurrent design generated by CODA could be compared against the student-generated design. Since CODA provides the specific decisions made and the rationale for those decisions, a student could learn in two ways. First, where the tool takes a more correct decision than the student, the student can identify and begin to understand her misconceptions about the CODARTS design method. Second, where the tool produces a poor design, the student can begin to understand the relationships between data/control flow diagrams and the heuristics included in the CODARTS design method.

12.3 Future Research

The research described in this dissertation leads to an automated representation of a concurrent design for real-time software. When generating the design, certain aspects of the intended target environment are considered, where appropriate. For example, the message queuing and software signaling services available in the target operating system are considered when defining task interfaces. Larger issues, such as the number of processors available in the target hardware configuration and the availability of various forms of shared memory, are not considered by the current research. In addition, various algorithms can be identified for assigning tasks in a design among processors in the hardware architecture and for assigning priorities to multiple tasks on a single processor. The current research does not use this information. All of these factors are included within the current research so that later research can address automated configuration of concurrent designs for specific hardware architectures.

Another area of future research involves evaluation of designs. The current research captures information about the frequency of task executions and about the maximum rate at which external stimuli arrive at the system. This information is not used within this dissertation but is intended to facilitate future research regarding automated evaluation of the performance of concurrent designs. One area for investigation is automated analysis of the schedulability of a concurrent design using rate-monotonic scheduling theory. This is a critical issue for real-time designs because a real-time system must meet its timing requirements event under worst-case system loading. A second area for investigation is dynamic simulation of concurrent designs. Dynamic simulation can be used to assess the average performance characteristics of a design under various loads, can be used to monitor the dynamic behavior of the design for undesirable properties, and can also be used to identify logical flaws in the design.

A third area for additional research is the generation of code skeletons from the internal representation of a concurrent design. The approach proposed in this dissertation leads to an automated representation for the entities and relationships composing a concurrent design. Perhaps a means can be found to generate code skeletons for the tasks and modules contained within the automated representation of the design.

A fourth area for additional research is automated analysis of state-transition diagrams to identify mutual exclusion among enabled functions and to identify locked-state events on a data/control flow diagram. Perhaps additional specification mechanisms could be proposed and investigated to show relationships among various external event flows and data flows. With this additional information, the ability to automatically analyze a data/control flow diagram for mutual exclusion and for locked-state events might be improved.

A fifth area for future research relates to the creation of data/control flow diagrams. At present, construction of a data/control flow diagram is left to a human designer. Researchers are currently investigating methods for extracting structured information from natural language requirements specifications. A gap exists between the requirements expressed as natural language and the requirements organized in a structured form such as data/control flow diagrams, state-transition diagrams, and data dictionaries. Today this gap must be filled by a human designer with little automated assistance. Future research might aim to provide automated assistance for constructing flow graphs from natural language requirements.

A sixth area for future research concerns the creation of a repository for design knowledge. As additional design heuristics are identified and existing heuristics are refined, an automated representation of the associated design rules can be specified, encoded, and maintained. In this way, an automated design assistant could increase its scope of knowledge and application over time. Additionally, partial designs for various applications might be stored in a repository and then serve as starting points for the generation of variants of the design when new target environments are required. This reuse of designs might prove particularly useful should the approach be extended to include the configuration and evaluation of designs.

A seventh area for future research involves extending the approach specified in this dissertation so that designs can be generated for distributed applications. The method proposed in this dissertation accommodates distributed applications by explicitly representing interfaces to external subsystems. A more comprehensive approach to distributed designs might prove useful, especially with the increasing importance of networks and client/server architectures.